UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/074,779	02/13/2002	Eric M. Dowling	MICS:0171-2	7948

52142 7590 01/21/2009

FLETCHER YODER (MICRON TECHNOLOGY, INC.)
P.O. BOX 692289
HOUSTON, TX 77269-2289

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
01/21/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/074,779	DOWLING, ERIC M.	
	Examiner	Art Unit	
	DAVID J. HUISMAN	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *22 October 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-14,16-32,34-47 and 49-51* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☒ Claim(s) *23-27 and 50* is/are allowed.

6)☒ Claim(s) *1-14,16-22,28-32,34-47 and 49-51* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *16 May 2005* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-14, 16-32, 34-47, and 49-51 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: Appeal Brief as received on 10/22/2008.

Reopening Prosecution

3. In view of the appeal brief filed on October 22, 2008, PROSECUTION IS HEREBY REOPENED. At least one new grounds of rejection set forth below in response to applicant's most recent amendment filed on February 15, 2008.

To avoid abandonment of the application, appellant must exercise one of the following two options:

(1) file a reply under 37 CFR 1.111 (if this Office action is non-final) or a reply under 37 CFR 1.113 (if this Office action is final); or,

(2) initiate a new appeal by filing a notice of appeal under 37 CFR 41.31 followed by an appeal brief under 37 CFR 41.37. The previously paid notice of appeal fee and appeal brief fee can be applied to the new appeal. If, however, the appeal fees set forth in 37 CFR 41.20 have been increased since they were previously paid, then appellant must pay the difference between the increased fees and the amount previously paid.

A Supervisory Patent Examiner (SPE) has approved of reopening prosecution by signing below.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1, 6-10, 12-14, and 51 are rejected under 35 U.S.C. 103(a) as being unpatentable over Inagami et al., U.S. Patent No. 4,881,168 (as applied in the previous Office Action and herein referred to as Inagami), in view of Luk et al., U.S. Patent No. 5,883,814 (herein referred to as Luk), and further in view of Farmwald et al., U.S. Patent No. 5,243,703 (herein referred to as Farmwald). In addition, Garcia, U.S. Patent No. 4,827,476, is cited as extrinsic evidence which shows that DRAM is inaccessible during refresh cycles.

6. Referring to claim 1, Inagami has taught a processor comprising:

a) an array comprising a plurality of random access memory cells arranged in rows and columns. See Fig.1 and note that main memory (component 1) inherently comprises rows and columns of memory cells. Inagami has not explicitly taught that the memory array is an embedded DRAM array (embedded on the same chip as the processor). However, Official Notice is taken that DRAM and its advantages are well known and accepted in the art. More specifically, DRAM is a very popular memory technology because of its high density (due to structural simplicity) and low price (in comparison to other memory such as SRAM). Furthermore, Luk has shown that a processor and DRAM can be embedded on a single chip. See Fig.2. Luk has further taught, in column 2, lines 1-14, that such an integration allows for the elimination of off-chip drivers and

heavy capacitive loads, low power dissipation, and less communication time between the two components. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to first modify Inagami's main storage to be DRAM, for the advantages described above, and to then embed the DRAM array on the same chip as the processor.

b) a row address register that holds a pointer that points to a row of the DRAM array. See Fig.1, component 60, and column 5, lines 13-17.

c) one or more sets of registers, each of said sets of registers capable of loading or storing an entire row of the DRAM array in response to a single latch signal. See Fig.1, Fig.4, Fig.5, and column 1, lines 52-64, and column 6, lines 48-53. Note that each 4-register set is capable of loading/storing an entire row of the DRAM array, regardless of how "row" is interpreted in Inagami, in response to a single load instruction (latch signal). That is, a single DRAM array row may be interpreted to correspond to a single entry shown in storage 21 in Fig.4. For instance, data a0 is in row 0, data a1 is in row 1, etc. By loading a0, for instance, as shown in Fig.4, the system loads the entire row that held a0. Or, in an alternate interpretation, a single DRAM row may be interpreted to include all of the a0 through a15 entries. In such a situation, the registers are still capable of loading the entire row when all of the mask bits 22 are '1'. As shown, in Fig.4, when a mask bit is '1', the corresponding row value is loaded into a register. When a mask bit is '0', the corresponding row value is not loaded into a register. Consequently, when all mask values are '1', the entire row will be loaded at once. It should be realized that such a mask would have been apparent to one of ordinary skill in the art.

d) an instruction set which includes:

(i) at least one command to perform an arithmetic operation on said row address register. See Fig.6, column 7, line 66, to column 8, line 9, and note that arithmetic is performed on the row address register by the start address calculation unit 310.

(ii) a command to load a plurality of words of a precharged row into designated sets of data registers. See Fig.4, and column 1, lines 52-59. Note also that in DRAM, the row must be precharged before it is read, as this is how DRAM technology functions.

(iii) a command to load selected columns of a precharged row pointed to by said row address register into designated sets of data registers, said selection based on bits in a mask, wherein all the selected columns of the precharged row are loaded into the designated sets of data registers in a single operation. Again, see Fig.4, the abstract, column 1, lines 52-59, and column 6, line 37, to column 7, line 8. Note that in a first interpretation, a0-a3 columns are loaded into the designated register set in response a single load instruction. And, in an alternate interpretation, each of the sets is loaded based on a mask 22 in response to a single parallel load operation, where the parallel load includes individual load instructions.

(iv) Inagami in view of Luk has not explicitly taught a command to precharge (activate) the row pointed to by said row address register. However, it is known that precharging of DRAM must occur. Farmwald has taught an instruction which explicitly precharges a row. See column 7, lines 49-55, and note that a bus transaction instruction is issued. It includes the fields shown in Fig.4. When bit 2 of AccessType is set to the appropriate value or when the 3rd bit of AccessType is set to the appropriate value, a row precharge will occur. See column 12, lines 34-53, and column 11, line 25-60. Note that such a

command allows for precharging ahead of time so that when an actual load occurs, for instance, the load can begin immediately, thereby saving time. Consequently, it would have been obvious to one of ordinary skill in the art to include a precharge command in the instruction set, as taught by Farmwald.

(v) Inagami in view of Luk has not explicitly taught a command to deactivate said row pointed to by said row address register after it had been precharged by the command to precharge. However, it is known that any load that occurs in a DRAM is a destructive load. That is, reading a value in DRAM causes the charges making up that value to diminish. Consequently, after a value is read, the DRAM row must be refreshed. Farmwald has also taught a command which performs an explicit refresh. See Fig.4 and column 12, lines 58-60. As is known in the art, a refresh deactivates the DRAM because, while it is being refreshed, no read/write requests can be serviced. See Garcia, column 1, line 66, to column 2, line 1 for support for this concept. As a result, in order to ensure that the DRAM doesn't lose its contents after a read, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to include a command to perform a refresh (i.e., a deactivate) of a particular row after it had been precharged (and read).

7. Referring to claim 6, Inagami in view of Luk and further in view of Farmwald has taught a processor as described in claim 1. Inagami has further taught a plurality of DRAM arrays. More specifically, the smallest array possible is either 1 row or 1 column. Clearly, Inagami has more rows and columns than this; otherwise, very little data would be stored. Consequently, Inagami would have many DRAM arrays (for instance, may 1x1 memory arrays).

8. Referring to claim 7, Inagami in view of Luk and further in view of Farmwald has taught a processor as described in claim 1. Inagami has further taught:

a) at least one functional unit. See Fig.1, component 5.

b) whereby said one or more sets of registers comprise a plurality of register files (see Fig.1, components Vro-VR7 and VMR0-VMR7), each of said register files comprising a parallel access port operative to load or store contents of said register file in a single cycle from or to a DRAM row as selected by said row-address register (again see Fig.1), each of said register files further comprising at least a second access port operative to transfer data between said functional unit and a selected subset register in said register file (note that the register files are coupled to the functional units 5 via switches).

9. Referring to claim 8, Inagami in view of Luk and further in view of Farmwald has taught a processor as described in claim 7. Inagami has further taught:

a) a second functional unit. See Fig.1, component 2, for instance.

b) whereby said first functional unit executes a first command to perform logical processing on the contents of one or more registers within a selected active one of said register sets (as seen in Fig.1, component 5 performs operations on register operands), and said second functional unit executes a second command to parallely transfer data between a selected inactive one of said register sets and said DRAM array (note that the load/store pipes do the loading and that a register file may be loaded while not being used to supply operands, i.e., inactive).

10. Referring to claim 9, Inagami in view of Luk and further in view of Farmwald has taught a processor as described in claim 8. Inagami has not explicitly taught that said first and second functional units execute said first and second commands substantially contemporaneously.

However, it should be noted that the load/store pipes and the operation pipes are completely separate (see Fig.1). That is, there is not a single unit which executes both instructions. Consequently, both the first and second commands could execute simultaneously and this would result in the most efficiency as two operations may be performed in less time than performing one after the other. As a result, it would have been obvious to execute the first and second commands simultaneously because Inagami's hardware supports it.

11. Referring to claim 10, Inagami in view of Luk and further in view of Farmwald has taught a processor as described in claim 8. Inagami has further taught:

a) a first software module comprising a set of data manipulation commands, said first software module executed by said first functional unit. It is inherent that a group of instructions exist which causes the manipulation of register data. These commands, which include multiplication, addition, etc. (column 4, lines 35-37), would be executed by the operation pipes 5 (Fig.1).

b) a second software module comprising a set of parallel data transfer commands, said second software module being executed by said second functional unit. See column 1, lines 52-64, and note that these transfer commands are executed by the second functional unit (load/store pipes). See column 4, lines 32-35.

c) whereby said second software module operates in support of said first software module to prefetch data from said DRAM array into one of said register files in advance of said data being needed by said first software module. Clearly, when load instructions are executed, they are executed to bring data into the memory so that subsequent instructions may use that data. This is prefetching in that the data is prefetched before the consumer instruction actually requires it.

12. Referring to claim 12, Inagami in view of Luk and further in view of Farmwald has taught a processor as described in claim 1. Inagami in view of Luk and further in view of Farmwald has further taught:

a) first and second sets of functional units (Fig.1, components 5 and 2 respectively), said first and second sets of functional units having respective first and second instruction subsets. Clearly, the first functional unit will execute instructions such as multiplication and addition (manipulation instructions) while the second functional unit executes instructions for data transfer. See Inagami, column 4, lines 32-37.

b) whereby the second instruction subset includes said commands (ii), (iii), (iv) and (v). Since commands (ii)-(v) deal with memory, they will be executed by component 2.

c) Inagami has not explicitly taught that command (i) is part of the first instruction subset. However, the row address register holds memory addresses and Official Notice is taken that memory addresses may be generated by arithmetic units. This would allow for different types of addressing modes to exist, such as indirect addressing. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami such that command (i) is part of the first instruction subset.

13. Referring to claim 13, Inagami has taught a processor comprising:

a) an array comprising a plurality of random access memory cells arranged in rows and columns. See Fig.1 and note that main memory (component 1) inherently comprises rows and columns of memory cells. Inagami has not explicitly taught that the memory array is an embedded DRAM array (embedded on the same chip as the processor). However, Official Notice is taken that DRAM and its advantages are well known and accepted in the art. More specifically, DRAM is

a very popular memory technology because of its high density (due to structural simplicity) and low price (in comparison to other memory such as SRAM). Furthermore, Luk has shown that a processor and DRAM can be embedded on a single chip. See Fig.2. Luk has further taught, in column 2, lines 1-14, that such an integration allows for the elimination of off-chip drivers and heavy capacitive loads, low power dissipation, and less communication time between the two components. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to first modify Inagami's main storage to be DRAM, for the advantages described above, and to then embed the DRAM array on the same chip as the processor.

b) a row address register that holds a pointer that points to a row of the DRAM array. See Fig.1, component 60, and column 5, lines 13-17.

c) one or more sets of data registers, each of said sets of data registers capable of loading or storing an entire row of the DRAM array in response to a single latch signal. See Fig.1, Fig.4, Fig.5, and column 1, lines 52-64, and column 6, lines 48-53. Note that each 4-register set is capable of loading/storing an entire row of the DRAM array, regardless of how "row" is interpreted in Inagami, in response to a single load instruction (latch signal). That is, a single DRAM array row may be interpreted to correspond to a single entry shown in storage 21 in Fig.4. For instance, data a0 is in row 0, data a1 is in row 1, etc. By loading a0, for instance, as shown in Fig.4, the system loads the entire row that held a0. Or, in an alternate interpretation, a single DRAM row may be interpreted to include all of the a0 through a15 entries. In such a situation, the registers are still capable of loading the entire row when all of the mask bits 22 are '1'. As shown, in Fig.4, when a mask bit is '1', the corresponding row value is loaded into a register. When a mask bit is '0', the corresponding row value is not loaded into a register.

Consequently, when all mask values are '1', the entire row will be loaded at once. It should be realized that such a mask would have been apparent to one of ordinary skill in the art.

d) a bit mask to select one or more data locations within at least one of said register sets. See Fig.4 and Fig.5, component 22.

e) an instruction set which comprises at least:

(i) a command to perform arithmetic operations on said row address register. See Fig.6, column 7, line 66, to column 8, line 9, and note that arithmetic is performed on the row address registers by the start address calculation unit 310.

(ii) a command to load a set of selected elements of the row pointed to by said row address register into a selected set of said data registers, said selection of elements based on bits in said mask, wherein all the selected elements of the precharged row are loaded into the selected set of data registers in a single operation. Again, see Fig.4, the abstract, column 1, lines 52-59, and column 6, line 37, to column 7, line 8. Note that in a first interpretation, a0-a3 columns are loaded into the selected register set in response a single load instruction.

(iii) Inagami in view of Luk has not explicitly taught a command to precharge (activate) the row pointed to by said row address register. However, it is known that precharging must occur. Farmwald has taught an instruction which explicitly precharges a row. See column 7, lines 49-55, and note that a bus transaction instruction is issued. It includes the fields shown in Fig.4. When bit 2 of AccessType is set to the appropriate value or when the 3rd bit of AccessType is set to the appropriate value, a row precharge will occur. See column 12, lines 34-53, and column 11, line 25-60. Note that such a command allows for

precharging ahead of time so that when an actual load occurs, for instance, the load can begin immediately, thereby saving time. Consequently, it would have been obvious to one of ordinary skill in the art to include a precharge command in the instruction set, as taught by Farmwald.

f) wherein the command to precharge is executed to precharge the row prior to the command to load so that at the time the command to load is issued, the command to load can execute without the need to wait for the row to precharge. See column 11, lines 25-34, of Farmwald.

14. Referring to claim 14, Inagami in view of Luk and further in view of Farmwald has taught a processor as described in claim 13. Inagami has further taught that said load command causes an entire row that was previously precharged to be loaded. Note from Fig.4 that if the mask were set to all 1's, then the entire row would be loaded.

15. Referring to claim 51, Inagami in view of Luk and further in view of Farmwald has taught a processor as described in claim 1. Inagami has further taught that each of said sets of registers is capable of being loaded or stored in response to a single latch signal without a caching system that employs cache hits and cache misses. Se Fig.1 and note that a cache is not disclosed anywhere in Inagami, and therefore, everything performed in Inagami is performed without a cache employing hits and misses.

16. Claims 2-5 and 11 are rejected under 35 U.S.C. 103(a) as being unpatentable over Inagami in view of Luk in view of Farmwald and further in view of Parady, U.S. Patent No. 5,933,627.

17. Referring to claim 2, Inagami in view of Luk in view of Farmwald has taught a processor as described in claim 1.

a) Inagami has further taught first and second sets of functional units, said first and second sets of functional units having respective first and second instruction sets and capable of accessing first and second sets of said register. See Fig.1, components 5-0 to 5-3, and note that these functional units may access any of the register sets.

b) Inagami has not taught a command to select one of said first and second sets of registers to be an architectural set of registers accessible to said first set of functional units, a command to deselect the other of said first and second sets of registers so that it is no longer an architectural register set accessible to said first set of functional units, a command to select one of said first and second sets of registers to be an architectural set of registers accessible to said second set of functional units, and a command to deselect the other one of said first and second sets of registers so that it is no longer an architectural register set accessible to said second set of functional units. However, Parady has taught a thread switch command which performs such operations. More specifically, in Parady, a first set of functional units may comprise components 38 and 40 in Fig.1 (which would execute a first set of instructions comprising floating-point addition, subtraction, and multiplication operations). In response to a thread switch command, a new register set (Fig.1, component 50) corresponding to the switched-in (current) thread will be made accessible to the first set of functional units. Meanwhile, the previously used register set corresponding to the switch-out (inactive) thread is no longer accessed by the first set of functional units. See the abstract and column 2, lines 18-39. Likewise, the thread switch command will also make the new register set available to a second set of functional units (for

example, components 42 and 44 in Fig.1, which would execute a second set of instructions comprising floating-point division and graphical addition and subtraction operations), whereas the previously used register set will become "invisible" to the second set of functional units. A person of ordinary skill in the art would have recognized that multithreading and thread switching are beneficial tools for a system because when one thread stalls, another thread may be switched in and executed, thereby preventing the processor from going idle. With thread switching comes the switching of register files as well. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to toggle between active and inactive register file states because switching of register sets supports thread switching, which as described above, prevents a processor from staying idle during a stall, thereby increasing throughput and efficiency.

18. Referring to claim 3, Inagami in view of Luk in view of Farmwald has taught a processor as described in claim 1.

a) Inagami has further taught first and second sets of functional units, said first and second sets of functional units having respective first and second instruction sets and capable of accessing first and second sets of said registers. See Fig.1, components 5-0 to 5-3, and note that these functional units may access any of the register sets.

b) Inagami has not taught a command which selects one of said first and second sets of registers to be an architectural set of registers accessible to said first set of functional units, and at the same time, which deselects the other one of said one of said first and second sets of registers to be an architectural set of registers accessible to said second set of functional units. However, Parady has taught a thread switch command which performs such operations. More specifically,

in Parady, a first set of functional units may comprise components 34 and 36 in Fig.1 (which would execute a first set of instructions comprising integer ALU, multiplication, and division operations). In response to a thread switch command, a new register set (Fig.1, component 48) corresponding to the switched-in (current) thread will be made accessible to the first set of functional units (integer registers for integer functional units). At the same time, the new register set will not be accessible to a second set of functional unit comprising components 38 and 40 in Fig.1 (which would execute floating-point addition, subtraction, and multiplication operations). This is because the selected register file is an integer register file and floating-point functional units would not access the integer register file (they would access a floating-point register file 50). A person of ordinary skill in the art would have recognized that multithreading and thread switching are beneficial tools for a system because when one thread stalls, another thread may be switched in and executed, thereby preventing the processor from going idle. With thread switching comes the switching of register files as well. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to toggle between active and inaccessible register file states because switching of register sets supports thread switching, which as described above, prevents a processor from staying idle during a stall, thereby increasing throughput and efficiency.

19. Referring to claim 4, Inagami in view of Luk in view of Farmwald has taught a processor as described in claim 1. Inagami has not explicitly taught the specifics of the functional units 5-0 to 5-3, and more specifically has not taught first and second sets of functional units, said first and second sets of functional units having respective first and second instruction sets and accessing first and second sets of said registers and whereby said first and second instruction sets are

subsets of said instruction set of said embedded-DRAM processor. However, Parady has taught the functional units may be of a floating-point type and of an integer type. See Fig.1. These two sets of functional units would execute integer and floating-point instructions, respectively, wherein each is a subset of the overall instruction set (integer + floating-point + miscellaneous instructions). Having different types of functional unit allows for the execution of different types of instructions and consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to include different types of functional units.

20. Referring to claim 5, Inagami in view of Luk in view of Farmwald and further in view of Parady has taught a processor as described in claim 4. Inagami has not explicitly taught that said second set of functional units comprises a functional unit that is a multi-issue functional unit and further comprises a dispatch unit and a plurality of functional units which each execute a respective instruction stream as dispatched by said dispatch unit. However, Parady has taught such a concept. Note that the second functional unit may comprise components 28, 38, 40, 42, 44, and 46 of Fig.1. This includes a dispatch unit, which must inherently exists to dispatch instructions (in this case 4 instructions at a time (column 3, lines 14-17)), and a plurality of functional units (38-46) which may each execute dispatched instructions. Having multiple functional units and a multi-issue dispatcher allows multiple instructions to be executed at any given time, thereby increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to have a multi-issue functional unit.

21. Referring to claim 11, Inagami in view of Luk in view of Farmwald has taught a processor as described in claim 10. Inagami has further taught that:

a) said first software module contains an instruction that reference registers within an architectural register set visible to said first functional unit, whereby said architectural register set corresponds to at least partially to said one of said register files that is in an active state. See column 4, lines 35-37, and note addition and multiplication instructions would exist which operate on data retrieved from register files (Fig.1). Clearly, if the functional unit 5 is retrieving data from a register file, then that register file is visible and in an active state.

b) said second software module contains instructions that cause data to be transferred between an inactive register set and said DRAM array. As discussed above, a register file does not need to be read every cycle. Therefore, it may be inactive with respect to the functional units. However, since the load pipes are separate from the functional units, the data transfer may occur even if the file is not being used by a functional unit.

c) Inagami has not taught that the second software module also executes a command to toggle a selected register set between said active and inactive states. However, Parady has taught a thread switch command which performs such toggling. More specifically, in Parady, every time a thread switch occurs, a new register file becomes active and the previously active register file becomes inactive. This allows for thread switching without having to save or reload a context of a thread, thereby saving processing time. A person of ordinary skill in the art would have recognized that multithreading and thread switching are beneficial tools for a system because when one thread stalls, another thread may be switched in and executed, thereby preventing the processor from going idle. With thread switching comes the switching of register files as well. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to toggle between active and inactive register file states because

switching of register sets supports thread switching, which as described above, prevents a processor from staying idle during a stall, thereby increasing throughput and efficiency.

22. Claims 16-22, 28-32 and 34-43, 45 and 49 are rejected under 35 U.S.C. 103(a) as being unpatentable over Inagami in view of Luk in view of Parady, and further in view of Bissett et al., U.S. Patent No. 5,896,523 (herein referred to as Bissett).

23. Referring to claim 16, Inagami has taught a processor comprising:

a) an array comprising a plurality of random access memory cells arranged in rows and columns. See Fig.1 and note that main memory (component 1) inherently comprises rows and columns of memory cells. Inagami has not explicitly taught that the memory array is an embedded DRAM array (embedded on the same chip as the processor). However, Official Notice is taken that DRAM and its advantages are well known and accepted in the art. More specifically, DRAM is a very popular memory technology because of its high density (due to structural simplicity) and low price (in comparison to other memory such as SRAM). Furthermore, Luk has shown that a processor and DRAM can be embedded on a single chip. See Fig.2. Luk has further taught, in column 2, lines 1-14, that such an integration allows for the elimination of off-chip drivers and heavy capacitive loads, low power dissipation, and less communication time between the two components. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to first modify Inagami's main storage to be DRAM, for the advantages described above, and to then embed the DRAM array on the same chip as the processor.

b) a row address register that holds a pointer that points to a row of the DRAM array. See Fig.1, component 60, and column 5, lines 13-17.

c) first and second register files, each of said register files having a plurality of data registers capable of loading or storing an entire row of the DRAM array in response to a single latch signal. See Fig.1, Fig.4, Fig.5, and column 1, lines 52-64, and column 6, lines 48-53. Note that each 4-register set is capable of loading/storing an entire row of the DRAM array, regardless of how "row" is interpreted in Inagami, in response to a single load instruction (latch signal). That is, a single DRAM array row may be interpreted to correspond to a single entry shown in storage 21 in Fig.4. For instance, data a0 is in row 0, data a1 is in row 1, etc. By loading a0, for instance, as shown in Fig.4, the system loads the entire row that held a0. Or, in an alternate interpretation, a single DRAM row may be interpreted to include all of the a0 through a15 entries. In such a situation, the registers are still capable of loading the entire row when all of the mask bits 22 are '1'. As shown, in Fig.4, when a mask bit is '1', the corresponding row value is loaded into a register. When a mask bit is '0', the corresponding row value is not loaded into a register. Consequently, when all mask values are '1', the entire row will be loaded at once. It should be realized that such a mask would have been apparent to one of ordinary skill in the art.
d) Inagami has not taught that each of said register files is also capable of being placed into an active state and an inactive state. However, Parady has taught a thread switch command which performs such state toggling. More specifically, in Parady, every time a thread switch occurs, a new register file becomes active and the previously active register file becomes inactive. This allows for thread switching without having to save or reload a context of a thread, thereby saving processing time. A person of ordinary skill in the art would have recognized that multithreading and thread switching are beneficial tools for a system because when one thread stalls, another thread may be switched in and executed, thereby preventing the processor from going idle. With

thread switching comes the switching of register files as well. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to toggle between active and inactive register file states because switching of register sets supports thread switching, which as described above, prevents a processor from staying idle during a stall, thereby increasing throughput and efficiency.

e) a set of functional units that perform logical operations on data accessed from a set of architectural registers, wherein registers placed into the active state appear as architectural registers to a set of functional units, and registers in the inactive state are not accessible by the functional units. See Fig.1 and Fig.3 of Parady. Note that multiple functional units (36, 38, 40, etc) would operate on instructions from the current thread, which has an associated active register file (Fig.3, files 48-50). Therefore, the active file is available to the functional units while the files for the inactive threads are not available to the units.

f) a bit mask to select one or more locations within at least one of said register files. See Fig.4 and Fig.5, component 22.

g) an instruction set which comprises at least:

(i) a command to perform an arithmetic operation on said row address register. See Fig.6, column 7, line 66, to column 8, line 9, and note that arithmetic is performed on the row address registers by the start address calculation unit 310.

(ii) a command to load a set of selected elements of the row pointed to by said row address register into a selected set of said data registers, said selection of elements based on bits in said mask, wherein all the selected elements of the row are loaded into the selected set of data registers in a single operation. Again, see Fig.4, the abstract, column

1, lines 52-59, and column 6, line 37, to column 7, line 8. Note that in a first interpretation, a0-a3 columns are loaded into the selected register set in response a single load instruction.

Inagami in view of Parady has not taught that the selected set of said data registers to be loaded are in the inactive state. However, Bissett has taught performing data prefetches (loads) in the background. And, this is beneficial because background operations do not influence software execution, thereby allowing the current program to run normally while also accomplishing a task in the background. See column 4, lines 3-8. And, as is known in the art, prefetching is beneficial because data is brought in from memory before an instruction needs it. Then, when the instruction does actually execute, the data is already fetched, allowing the instruction to execute more quickly. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami in view of Parady in view of Bissett such that Inagami in view of Parady performs background prefetching to a register file is not currently in use (one that corresponds to an inactive thread).

24. Referring to claim 17, Inagami in view of Luck in view of Parady and further in view of Bissett has taught a processor as described in claim 16. Parady has further taught that the instruction set further comprises a command to toggle a register set between said active and inactive states. As discussed in Parady's abstract, a thread switch occurs on a cache miss during a load instruction's execution. Therefore, this command is built into a load instruction which is part of the instruction set.

25. Referring to claim 18, Inagami in view of Luck in view of Parady and further in view of Bissett has taught a processor as described in claim 17. Parady has further taught that said toggle command causes said first register file to toggle from the inactive state to the active state and also causes the second register file to toggle from the active state to the inactive state. Again, as described in the rejection of claim 16 above, every time a thread switch occurs, a new register file becomes active and the previously active register file becomes inactive. That is, each thread has its own register file (column 2, lines 35-37). Consequently, when a thread switch occurs, the register file corresponding to the switched in thread goes from inactive to active, while the register file corresponding to the switched out thread goes from active to inactive.

26. Referring to claim 19, Inagami in view of Luck in view of Parady and further in view of Bissett has taught a processor as described in claim 16. Inagami has further taught that the instruction set further comprises a command to manipulate the bits in the bit mask. Although the command is not explicitly mentioned, the vector masks are stored in register files VMR0-VMR7 and they may be written to (modified) as seen in Fig.1.

27. Referring to claim 20, Inagami in view of Luck in view of Parady and further in view of Bissett has taught a processor as described in claim 16. Inagami has further taught:
a) first and second sets of functional units, said first and second sets of functional units having respective first and second sets of instructions and capable of accessing said first and second register sets. See Fig.1 and note that the first set of functional units could be component 5, which executes manipulation instructions like multiplication, addition, etc. See column 4, lines 35-37. The second set of functional units could be component 2 of Fig.1, which executes data transfer instructions such as loads and stores. See column 4, lines 32-35.

b) said instruction set further comprises at least:

(i) a command to select one of said first and second sets of registers to be an architectural set of registers accessible to said first set of functional units. See Fig.1. Clearly, if a functional unit (5-0, for instance) executes an addition instruction, it will select at least one of the register files so that it may access operands.

(ii) a command to select one of said first and second sets of registers to be an architectural set of registers accessible to said second set of functional units. See Fig.1. Clearly, if a functional unit (2-0, for instance) executes a store instruction, it will select at least one of the register files so that it may access an operand to be stored.

28. Referring to claim 21, Inagami in view of Luck in view of Parady and further in view of Bissett has taught a processor as described in claim 20. Inagami has not taught that the instruction set further comprises a command to deselect the other of said first and second sets of registers so that it is no longer an architectural register set accessible to said first set of functional units and a command to deselect the other one of said first and second sets of registers so that it is no longer an architectural register set accessible to said second set of functional units. However, Parady has taught such a concept. More specifically, in Parady, every time a thread switch occurs, a new register file becomes active and the previously active register file becomes inactive (deselected so it is no longer accessible). This allows for thread switching without having to save or reload a context of a thread, thereby saving processing time. A person of ordinary skill in the art would have recognized that multithreading and thread switching are beneficial tools for a system because when one thread stalls, another thread may be switched in and executed, thereby preventing the processor from going idle. With thread switching comes

the switching of register files as well. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to toggle between selection and deselection of register files because switching of register sets supports thread switching, which as described above, prevents a processor from staying idle during a stall, thereby increasing throughput and efficiency.

29. Referring to claim 22, Inagami in view of Luck in view of Parady and further in view of Bissett has taught a processor as described in claim 20. Inagami has further taught that at least one of said sets of functional units contains a single functional unit. See Fig.1, component 5, and note that it contains a single unit 5-0.

30. Referring to claim 28, Inagami has taught a processor comprising:

a) an array comprising a plurality of random access memory cells arranged in rows and columns. See Fig.1 and note that main memory (component 1) inherently comprises rows and columns of memory cells. Inagami has not explicitly taught that the memory array is an embedded DRAM array (embedded on the same chip as the processor). However, Official Notice is taken that DRAM and its advantages are well known and accepted in the art. More specifically, DRAM is a very popular memory technology because of its high density (due to structural simplicity) and low price (in comparison to other memory such as SRAM). Furthermore, Luk has shown that a processor and DRAM can be embedded on a single chip. See Fig.2. Luk has further taught, in column 2, lines 1-14, that such an integration allows for the elimination of off-chip drivers and heavy capacitive loads, low power dissipation, and less communication time between the two components. Consequently, it would have been obvious to one of ordinary skill in the art at the

time of the invention to first modify Inagami's main storage to be DRAM, for the advantages described above, and to then embed the DRAM array on the same chip as the processor.

b) first and second dual-port register files, each of said register files capable of parallely transferring data between an entire row of said DRAM array in a single operation. See Fig.1, Fig.4, Fig.5, and column 1, lines 52-64, and column 6, lines 48-53. Note that each 4-register set is capable of loading/storing an entire row of the DRAM array, regardless of how "row" is interpreted in Inagami, in response to a single load instruction (latch signal). That is, a single DRAM array row may be interpreted to correspond to a single entry shown in storage 21 in Fig.4. For instance, data a0 is in row 0, data a1 is in row 1, etc. By loading a0, for instance, as shown in Fig.4, the system loads the entire row that held a0. Or, in an alternate interpretation, a single DRAM row may be interpreted to include all of the a0 through a15 entries. In such a situation, the registers are still capable of loading the entire row when all of the mask bits 22 are '1'. As shown, in Fig.4, when a mask bit is '1', the corresponding row value is loaded into a register. When a mask bit is '0', the corresponding row value is not loaded into a register. Consequently, when all mask values are '1', the entire row will be loaded at once. It should be realized that such a mask would have been apparent to one of ordinary skill in the art. Also, it should be noted that these files are at least dual-port in that they have ports for reading and writing (see Fig.1)

c) Inagami has not taught that each of said register files is also capable of being placed into an active state and an inactive state. However, Parady has taught a thread switch command which performs such state toggling. More specifically, in Parady, every time a thread switch occurs, a new register file becomes active and the previously active register file becomes inactive. This

allows for thread switching without having to save or reload a context of a thread, thereby saving processing time. A person of ordinary skill in the art would have recognized that multithreading and thread switching are beneficial tools for a system because when one thread stalls, another thread may be switched in and executed, thereby preventing the processor from going idle. With thread switching comes the switching of register files as well. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to toggle between active and inactive register file states because switching of register sets supports thread switching, which as described above, prevents a processor from staying idle during a stall, thereby increasing throughput and efficiency.

d) first and second embedded functional units, said first and second functional units having respective first and second instruction sets that operate on registers of an architectural register set, said architectural register set comprising one of the first and second dual-port register files that is currently in the active state. See Fig.1 and note that the first set of functional units would be component 5, which executes manipulation instructions like multiplication, addition, etc. See column 4, lines 35-37. The second set of functional units would be component 2 of Fig.1, which executes data transfer instructions such as loads and stores. See column 4, lines 32-35. Also, as taught in Parady, only the register files in the active state would be accessible.

e) whereby said first instruction set comprises at least:

(i) a command to manipulate data in a data register with a register file. See column 4, lines 35-37.

f) whereby said second instruction set comprises at least:

(i) a command to place an said selected inactive data register file into said active state, wherein when the inactive register file is activated, it becomes an architectural register set of said first functional unit. See Parady, and note the thread switch command that from the abstract and from Fig.3. More specifically, when a thread switch occurs, the register file associated with the "switched-in" thread will become active and consequently used by the system during execution of that thread.

(ii) Inagami has taught a command to unidirectionally transfer data between an entire row of said DRAM array and a data register file, wherein the transfer occurs in a single operation. See Fig.4. Neither Inagami nor Parady have taught such a transfer involves an inactive register file. However, Bissett has taught performing data prefetches (loads) in the background. And, this is beneficial because background operations do not influence software execution, thereby allowing the current program to run normally while also accomplishing a task in the background. See column 4, lines 3-8. And, as is known in the art, prefetching is beneficial because data is brought in from memory before an instruction needs it. Then, when the instruction does actually execute, the data is already fetched, allowing the instruction to execute more quickly. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami in view of Parady in view of Bissett such that Inagami in view of Parady performs background prefetching to a register file is not currently in use (one that corresponds to an inactive thread).

(iii) ***NOTE*** As an alternate interpretation, an inactive register file could be nothing more than a file which is not currently being accessed by the functional units. This could

be the case in Fig.1, where there are many register files. Some may not be needed by functional units 5. These would be inactive and a load instruction may be the instructions used to transfer data to the inactive file. In addition, when the functional units do request operands from the inactive file (say in response to an add instruction), then the inactive file becomes active and accessible to the functional unit. Applicant should amend accordingly to avoid both interpretations of Inagami, etc.

31. Referring to claim 29, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami has further taught that said command to unidirectionally transfer data causes data to be transferred from a row of the DRAM array to said selected inactive data register file. See Fig.4.

32. Referring to claim 30, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami has further taught that said command to unidirectionally transfer data causes data to be transferred from said selected inactive data register file to a row of the DRAM array. See Fig.5.

33. Referring to claim 31, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Parady has further taught that the said command to place the selected inactive register file into the active state is a command that also causes the remaining register file to toggle from the active state into the inactive state. As discussed in Parady's abstract, a thread switch occurs on a cache miss during a load instruction's execution. Therefore, this command is built into a load instruction which is part of the instruction set.

34. Referring to claim 32, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28.

a) Inagami has further taught at least one additional register file. Note that there are at least 16 register files shown in Fig.1 (VMR0-VMR7 and VR0-VR7).

b) Parady has further taught that said command to place the selected inactive register file into the active state is a command that also causes a selected other register file to toggle from the active state into the inactive state. As discussed in Parady's abstract, a thread switch occurs on a cache miss during a load instruction's execution. Therefore, this command is built into a load instruction which is part of the instruction set.

35. Referring to claim 34, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami has further taught:

a) at least one bit mask. See Fig.4, component 22.

b) the second instruction set further comprises a command to move a subset of elements between a selected register file and a selected row of said DRAM array, whereby said subset is identified by said bit mask. See fig.4 and Fig.5.

36. Referring to claim 35, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28.

a) Inagami has not explicitly taught a dispatch unit but it is deemed inherent by the examiner that such a component exists. That is, instructions must be dispatched somehow to the functional units. Consequently, a dispatch unit is required.

b) Inagami has further taught a plurality of functional units that each execute a respective instruction stream as dispatched by said dispatch unit. See functional units 5-0 to 5-3 in Fig.1. Each of units 5 may execute mult, add, etc., all of which must be dispatched.

c) Inagami has not explicitly taught that the first functional unit is a multi-issue functional unit. However, Parady has taught such a concept. Note that the functional unit of Parady may comprise components 28, 38, 40, 42, 44, and 46 of Fig.1. This includes a dispatch unit, which must inherently exists to dispatch instructions (in this case 4 instructions at a time (column 3, lines 14-17)), and a plurality of functional units (38-46) which may each execute dispatched instructions. Having multiple functional units and a multi-issue dispatcher allows multiple instructions to be executed at any given time, thereby increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to have a multi-issue functional unit.

37. Referring to claim 36, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami has further taught:

a) a first software module comprising a set of data manipulation commands drawn from said first instruction set, said first software module executed by said first functional unit. It is inherent that a group of instructions exist which causes the manipulation of register data. These commands, which include multiplication, addition, etc. (column 4, lines 35-37), would be executed by the operation pipes 5 (Fig.1).

b) a second software module comprising a set of parallel data transfer commands drawn from said second instruction set, said second software module being executed by said second

functional unit. See column 1, lines 52-64, and note that these transfer commands are executed by the second functional unit (load/store pipes). See column 4, lines 32-35.

c) whereby said second software module operates in support of said first software module to prefetch data from said DRAM array into one of said register files in advance of said data being needed by said first software module. Clearly, when load instructions are executed, they are executed to bring data into the memory so that subsequent instructions may use that data. This is prefetching in that the data is prefetched before the consumer instruction actually requires it.

38. Referring to claim 37, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami has further taught that:

a) said first software module contains an instruction that references registers within an architectural register set visible to said first functional unit, whereby said architectural register set corresponds to at least partially to said one of said register files that is in an active state. See column 4, lines 35-37, and note addition and multiplication instructions would exist which operate on data retrieved from register files (Fig.1). Clearly, if the functional unit 5 is retrieving data from a register file, then that register file is visible and in an active state.

b) said second software module contains instructions that cause data to be transferred between an inactive register set and said DRAM array. As discussed above, a register file does not need to be read every cycle. Therefore, it may be inactive with respect to the functional units. However, since the load pipes are separate from the functional units, the data transfer may occur even if the file is not being used by a functional unit.

c) Finally, recall that Parady has taught that the second software module also executes a command to toggle a selected register set between said active and inactive states. More

specifically, Parady has taught a thread switch command which performs such toggling. Every time a thread switch occurs, a new register file becomes active and the previously active register file becomes inactive.

39. Referring to claim 38, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami has further taught that each of said register files contain a number of words, N, matched to the number of words in a row of said DRAM array, and said unidirectional transfer comprises moving said selected row in its entirety to said selected register file. See Fig.4 and note that if the vector is all 1's then all of the rows will be moved to all of the registers. That is the register file may accommodate N words from DRAM.

40. Referring to claim 39, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami has further taught a mask and switch unit interposed between said DRAM array and at least one of said register files. See Fig.4 and Fig.5.

41. Referring to claim 40, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami in view of Luk in view of Parady and further in view of Bissett has not explicitly taught that said second set of instructions comprises a command to cause data to be moved from one register to another within a given one of said register files (individual register-to-register move operations). However, Official Notice is taken the register-register move operations are well known and expected in the art. These operations at the very least allow for the copying of registers. Consequently, it would have been

obvious to one of ordinary skill in the art at the time of the invention to implement register-register move operations.

42. Referring to claim 41, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28. Inagami has further taught that said second instruction set is used to implement an intelligent caching scheme, whereby said register files act as a cache and said second set of instructions are executed in lieu of a standard cache that maintains most recently used data and enforces a set associative or a direct-mapped caching policy. It should be noted that caches are not discussed in any way whatsoever in Inagami. Consequently, it is determined that Inagami does not employ a cache. And, register files hold recent operand data which is to be used by the processor for operations. As a result, the register files act as a cache for holding recent operation data.

43. Referring to claim 42, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 28.

a) Inagami has not explicitly taught an instruction register coupled to receive instructions from said instruction set, said instruction register operative to hold an instruction to be executed by a data assembly unit. However, the examiner has deemed an instruction register (IR) as being a component which inherently exists within a system that executes instructions. That is, when instructions are fetched from memory, they must be held in an IR so that they may be decoded and executed. This includes instructions to be executed by the data assembly unit (Fig.1, component 2).

b) Inagami has not explicitly taught a local program memory, but this is also inherent as instructions must be stored in some form of memory if they are to be executed.

c) Inagami has not taught that said second functional unit corresponds to said data assembly unit, and said data assembly unit receives an instruction from said second instruction set that causes a separate control thread of instructions to be accessed from said local program memory and executed by said data assembly unit. However, Parady has taught this concept. As discussed above, when a load is executed (by the data assembly unit) but it misses the cache, a thread switch occurs, thereby causing a new thread of instructions to be accessed and executed.

44. Referring to claim 43, Inagami in view of Luk in view of Parady and further in view of Bissett has taught a processor as described in claim 42.

a) Inagami has not explicitly taught a prefetch unit that prefetches instructions from the first and second instruction sets from a single very long instruction word (VLIW) instruction memory. However, Official Notice is taken that both prefetching and VLIW instructions are well known concepts in the art. Prefetching allows instructions/data to be fetched before they are actually needed. Therefore, when they are needed, they will be available immediately, as opposed to fetching them at that time. This will increase efficiency. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to include a prefetch unit. Also, VLIW instructions allow for multiple instructions to be executed at the same time and since they are bundled together during compilation, the need for dynamic scheduling is reduced. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to implement VLIW instructions.

b) Inagami has not explicitly taught a dispatch unit that dispatches instructions from the first instruction set to the functional units and dispatches instructions from the second instruction stream to the data assembly unit. However, such a component is deemed as inherently existing

by the examiner. That is, instructions must be dispatched somehow to the functional units. Consequently, a dispatch unit is required.

45. Referring to claim 45, Inagami has taught a processor comprising:

a) an array comprising a plurality of random access memory cells arranged in rows and columns. See Fig.1 and note that main memory (component 1) inherently comprises rows and columns of memory cells. Inagami has not explicitly taught that the memory array is an embedded DRAM array (embedded on the same chip as the processor). However, Official Notice is taken that DRAM and its advantages are well known and accepted in the art. More specifically, DRAM is a very popular memory technology because of its high density (due to structural simplicity) and low price (in comparison to other memory such as SRAM). Furthermore, Luk has shown that a processor and DRAM can be embedded on a single chip. See Fig.2. Luk has further taught, in column 2, lines 1-14, that such an integration allows for the elimination of off-chip drivers and heavy capacitive loads, low power dissipation, and less communication time between the two components. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to first modify Inagami's main storage to be DRAM, for the advantages described above, and to then embed the DRAM array on the same chip as the processor.

b) first and second dual-port register files (see Fig.1, components VMR0-VMR7 and VR0-VR7), whereby the first port of each of said register files is a parallel access port and is parallely coupled to said DRAM array (see Fig.1, note that if transfers occur between DRAM and registers, as shown in Fig.4 and Fig.5, then a port must couple registers to DRAM).

c) Inagami has not taught that each of said register files is capable of being placed into an active state and an inactive state. However, Parady has taught a thread switch command which

performs such state toggling. More specifically, in Parady, every time a thread switch occurs, a new register file becomes active and the previously active register file becomes inactive. This allows for thread switching without having to save or reload a context of a thread, thereby saving processing time. A person of ordinary skill in the art would have recognized that multithreading and thread switching are beneficial tools for a system because when one thread stalls, another thread may be switched in and executed, thereby preventing the processor from going idle. With thread switching comes the switching of register files as well. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to toggle between active and inactive register file states because switching of register sets supports thread switching, which as described above, prevents a processor from staying idle during a stall, thereby increasing throughput and efficiency.

d) at least one functional unit that executes a first program, said functional unit coupled to said second port of said register files, said functional unit responsive to commands exclusively involving architectural register operands that map onto to the registers within a register file that is in the active state. See Fig.1 and note that the first set of functional units would be component 5, which executes manipulation instructions like multiplication, addition, etc. In order to do this, it must be coupled to registers, as shown in Fig.1, for retrieving an operating upon operands.

e) a data assembly unit responsive to an instruction set comprising at least:

(i) a command that causes data to be moved between the DRAM array and a register file that is in the inactive state. See Parady, and note the thread switch command that from the abstract and from Fig.3. More specifically, when a thread switch occurs, the register file associated with the "switched-in" thread will become active and consequently used

by the system during execution of that thread. At the same time, the register file associated with the "switched-out" thread will become inactive.

(ii) Inagami has taught a command to transfer data between the DRAM array and a data register file. See Fig.4. Neither Inagami nor Parady have taught such a transfer involves an inactive register file. However, Bissett has taught performing data prefetches (loads) in the background. And, this is beneficial because background operations do not influence software execution, thereby allowing the current program to run normally while also accomplishing a task in the background. See column 4, lines 3-8. And, as is known in the art, prefetching is beneficial because data is brought in from memory before an instruction needs it. Then, when the instruction does actually execute, the data is already fetched, allowing the instruction to execute more quickly. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami in view of Parady in view of Bissett such that Inagami in view of Parady performs background prefetching to a register file is not currently in use (one that corresponds to an inactive thread).

(iii) ***NOTE*** As an alternate interpretation, an inactive register file could be nothing more than a file which is not currently being accessed by the functional units. This could be the case in Fig.1, where there are many register files. Some may not be needed by functional units 5. These would be inactive and a load instruction may be the instructions used to transfer data to the inactive file. In addition, when the functional units do request operands from the inactive file (say in response to an add instruction), then the inactive

file becomes active and accessible to the functional unit. Applicant should amend accordingly to avoid both interpretations of Inagami, etc.

46. Referring to claim 49, Inagami has taught a digital processor comprising:

a) an array comprising a plurality of random access memory cells arranged in rows and columns. See Fig.1 and note that main memory (component 1) inherently comprises rows and columns of memory cells. Inagami has not explicitly taught that the memory array is an embedded DRAM array (embedded on the same chip as the processor). However, Official Notice is taken that DRAM and its advantages are well known and accepted in the art. More specifically, DRAM is a very popular memory technology because of its high density (due to structural simplicity) and low price (in comparison to other memory such as SRAM). Furthermore, Luk has shown that a processor and DRAM can be embedded on a single chip. See Fig.2. Luk has further taught, in column 2, lines 1-14, that such an integration allows for the elimination of off-chip drivers and heavy capacitive loads, low power dissipation, and less communication time between the two components. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to first modify Inagami's main storage to be DRAM, for the advantages described above, and to then embed the DRAM array on the same chip as the processor.

b) first and second dual-port register files, each capable of parallel transfer of data between an entire row of said embedded DRAM array in a single operation. See Fig.1, Fig.4, Fig.5, and column 1, lines 52-64, and column 6, lines 48-53. Note that each 4-register set is capable of loading/storing an entire row of the DRAM array, regardless of how "row" is interpreted in Inagami, in response to a single load instruction (latch signal). That is, a single DRAM array row may be interpreted to correspond to a single entry shown in storage 21 in Fig.4. For

instance, data a0 is in row 0, data a1 is in row 1, etc. By loading a0, for instance, as shown in Fig.4, the system loads the entire row that held a0. Or, in an alternate interpretation, a single DRAM row may be interpreted to include all of the a0 through a15 entries. In such a situation, the registers are still capable of loading the entire row when all of the mask bits 22 are '1'. As shown, in Fig.4, when a mask bit is '1', the corresponding row value is loaded into a register. When a mask bit is '0', the corresponding row value is not loaded into a register. Consequently, when all mask values are '1', the entire row will be loaded at once. It should be realized that such a mask would have been apparent to one of ordinary skill in the art. Also, it should be noted that these files are at least dual-port in that they have ports for reading and writing (see Fig.1).

c) Inagami has not taught that each of said register files is also capable of being placed into an active state and an inactive state. However, Parady has taught a thread switch command which performs such state toggling. More specifically, in Parady, every time a thread switch occurs, a new register file becomes active and the previously active register file becomes inactive. This allows for thread switching without having to save or reload a context of a thread, thereby saving processing time. A person of ordinary skill in the art would have recognized that multithreading and thread switching are beneficial tools for a system because when one thread stalls, another thread may be switched in and executed, thereby preventing the processor from going idle. With thread switching comes the switching of register files as well. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to toggle between active and inactive register file states because switching of register sets supports thread switching, which as described above, prevents a processor from staying idle during a stall, thereby increasing throughput and efficiency.

d) first and second functional units. See Fig.1, components 5 and 2, respectively.

e) a method for processing data comprising:

(i) manipulating data in a data register with a register file using said first functional unit that is in an active state. See column 4, lines 35-37.

(ii) using said second functional unit. See column 4, lines 32-35.

(iii) placing an inactive register file into said active state, whereby when the register set is activated, it becomes an architectural register set of said first functional unit. See Parady, and note the thread switch command that from the abstract and from Fig.3. More specifically, when a thread switch occurs, the register file associated with the "switched-in" thread will become active and consequently used by the system during execution of that thread.

(ii) Inagami has taught unidirectionally transferring data between an entire row of said embedded DRAM array and a selected data register file, wherein the transfer of the data occurs in a single operation. See Fig.4. Neither Inagami nor Parady have taught such a transfer involves an inactive register file. However, Bissett has taught performing data prefetches (loads) in the background. And, this is beneficial because background operations do not influence software execution, thereby allowing the current program to run normally while also accomplishing a task in the background. See column 4, lines 3-8. And, as is known in the art, prefetching is beneficial because data is brought in from memory before an instruction needs it. Then, when the instruction does actually execute, the data is already fetched, allowing the instruction to execute more quickly. Consequently, it would have been obvious to one of ordinary skill in the art at the time of

the invention to modify Inagami in view of Parady in view of Bissett such that Inagami in view of Parady performs background prefetching to a register file is not currently in use (one that corresponds to an inactive thread).

(iii) ***NOTE*** As an alternate interpretation, an inactive register file could be nothing more than a file which is not currently being accessed by the functional units. This could be the case in Fig.1, where there are many register files. Some may not be needed by functional units 5. These would be inactive and a load instruction may be the instructions used to transfer data to the inactive file. In addition, when the functional units do request operands from the inactive file (say in response to an add instruction), then the inactive file becomes active and accessible to the functional unit. Applicant should amend accordingly to avoid both interpretations of Inagami, etc.

47. Claim 44 is rejected under 35 U.S.C. 103(a) as being unpatentable over Inagami in view of Luk in view of Parady in view of Bissett and further in view of Farmwald.

48. Referring to claim 44, Inagami in view of Luk in view of Parady in view of Bissett has taught a processor as described in claim 28.

a) Inagami in view of Luk in view of Parady and further in view of Bissett has not explicitly taught a command to precharge a row of the DRAM array, whereby the second functional unit executes a speculative precharging to prevent program delays due to DRAM row precharging. However, it is known that precharging must occur in a DRAM. Farmwald has taught an instruction which explicitly speculatively precharges a row. See column 7, lines 49-55, and note that a bus transaction instruction is issued. It includes the fields shown in Fig.4. When bit 2 of

AccessType is set to the appropriate value or when the 3rd bit of AccessType is set to the appropriate value, a row precharge will occur. See column 12, lines 34-53, and column 11, line 25-60. Note that such a command allows for precharging ahead of time so that when an actual load occurs, for instance, the load can begin immediately, thereby saving time. Consequently, it would have been obvious to one of ordinary skill in the art to include a speculative precharge command in the instruction set of Inagami, as taught by Farmwald.

49. Claim 46 is rejected under 35 U.S.C. 103(a) as being unpatentable over Inagami in view of Luk and further in view of Krishnamohan et al., U.S. Patent No. 5,499,355 (herein referred to as Krishnamohan).

50. Referring to claim 46, Inagami has taught a digital processor comprising:

a) an array comprising a plurality of random access memory cells arranged in rows and columns. See Fig.1 and note that main memory (component 1) inherently comprises rows and columns of memory cells. Inagami has not explicitly taught that the memory array is an embedded DRAM array (embedded on the same chip as the processor). However, Official Notice is taken that DRAM and its advantages are well known and accepted in the art. More specifically, DRAM is a very popular memory technology because of its high density (due to structural simplicity) and low price (in comparison to other memory such as SRAM). Furthermore, Luk has shown that a processor and DRAM can be embedded on a single chip. See Fig.2. Luk has further taught, in column 2, lines 1-14, that such an integration allows for the elimination of off-chip drivers and heavy capacitive loads, low power dissipation, and less communication time between the two components. Consequently, it would have been obvious to one of ordinary skill in the art at the

time of the invention to first modify Inagami's main storage to be DRAM, for the advantages described above, and to then embed the DRAM array on the same chip as the processor.

b) at least one row address register. See Fig.1, component 60, and column 5, lines 13-17.

c) one or more sets of registers capable of being loaded or stored. See Fig.1, components VMR0-VMR7 and VR0-VR7. Note that these registers may be loaded or stored in response to a single load or store signal shown in Fig.4, Fig.5, and column 1, lines 52-64.

d) a method of processing data comprising:

(i) performing an arithmetic operation on said at least one row address register in order to manipulate a pointer that points to a row of the embedded DRAM array. See Fig.6, column 7, line 66, to column 8, line 9, and note that arithmetic is performed on the row address registers by the start address calculation unit 310.

(ii) Inagami in view of Luk has not explicitly taught speculatively precharging (activating) a row pointed to by said at least one row address register based at least partially upon historical program execution data indicating a possible need to perform one or more load or store operations that would access said row. However, Krishnamohan has taught a predictor which predicts that a load will possibly occur at some point (based on historical PC values stored in the predictor) and as a result, performs a prefetch (i.e., speculative fetch). See Fig.7 and column 7, lines 20-64. This is clearly advantageous because, as is known in the art, fetching from main memory (DRAM) is time costly. By loading the data into temporary storage before the data needs to be actually loaded, at least some of the main memory access time is masked, thereby speeding up the system. As a result, it would have been obvious to one of ordinary skill

in the art at the time of the invention to modify Inagami to speculatively fetch, from memory, based at least partially on historical program execution data indicating a possible need to perform a load that would access the address speculatively fetched from. It follows that since a DRAM row must be precharged in order to fetch from it, it must be speculatively precharged prior to speculatively fetching from it.

(iii) Inagami in view of Luk and further in view of Krishnamohan has that that in response to a separate command executed after the speculatively precharging, loading a plurality of words of a row designated by said at least one row address register into designated sets of data registers in a single operation. See Fig.4, and column 1, lines 52-59, of Inagami.

51. Claim 47 is rejected under 35 U.S.C. 103(a) as being unpatentable over Inagami in view of Luk in view of Krishnamohan and further in view of Farmwald. In addition, Garcia, U.S. Patent No. 4,827,476, is cited as extrinsic evidence which shows that DRAM is inaccessible during refresh cycles.

52. Referring to claim 47, Inagami in view of Luk and further in view of Krishnamohan has taught a method as described in claim 46. Inagami in view of Luk has not explicitly taught deactivating rows pointed to by said at least one row address register. However, it is known that any load that occurs in a DRAM is a destructive load. That is, reading a value in DRAM causes the charges making up that value to diminish. Consequently, after a value is read, the DRAM row must be refreshed. Farmwald has taught a command which performs an explicit refresh. See Fig.4 and column 12, lines 58-60. As is known in the art, a refresh deactivates the DRAM

because, while it is being refreshed, no read/write requests can be serviced. See Garcia, column 1, line 66, to column 2, line 1 for support for this concept. As a result, in order to ensure that the DRAM doesn't lose its contents after a read, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Inagami to include Farmwald's command to perform a refresh (i.e., a deactivate) of a particular row after it has been precharged (and read).

Allowable Subject Matter

53. Claims 23-27 and 50 are allowed.

Response to Arguments

54. Applicant's arguments from the appeal brief filed on October 22, 2008, have been fully considered. In response, the rejections for claims 23-27 and 50 have been withdrawn. The remaining rejections are maintained for reasons set forth below.

55. Regarding applicant's argument that an entire row of DRAM is not loaded/store din a single operation, the examiner respectfully and strongly disagrees for the reasons set forth in the rejections above. Specifically, see Fig.1, Fig.4, Fig.5, and column 1, lines 52-64, and column 6, lines 48-53. Note that each register set is capable of loading/storing an entire row of the DRAM array, regardless of how "row" is interpreted in Inagami, in response to a single load instruction (latch signal). That is, a single DRAM array row may be interpreted to correspond to a single entry shown in storage 21 in Fig.4. For instance, data a0 is in row 0, data a1 is in row 1, etc. By loading a0, for instance, as shown in Fig.4, the system loads the entire row that held a0. Or, in an alternate interpretation, a single DRAM row may be interpreted to include all of the a0

through a15 entries. In such a situation, the registers are still capable of loading the entire row when all of the mask bits 22 are '1'. As shown, in Fig.4, when a mask bit is '1', the corresponding row value is loaded into a register. When a mask bit is '0', the corresponding row value is not loaded into a register. Consequently, when all mask values are '1', the entire row will be loaded at once. It should be realized that such a mask would have been apparent to one of ordinary skill in the art and that the specific mask values shown in Fig.4 are merely exemplary.

56. Regarding applicant's argument that the dual-port register file of claim 45 has not been taught by Inagami, the examiner strongly disagrees for the reasons set forth in the rejection of claim 45 above. That is, claim 45 merely requires that the first port of the register files be parallely coupled to a DRAM array while a second port be coupled to a functional unit. Looking at Fig.1 of Inagami, it can be seen that register files VR0-VR7 have one port coupled to the DRAM array 1 through switches 120-123 and load/store pipes 2, and a second port coupled to a functional unit 5 through switches 124-127. Note that this claimed dual-port register file is different from that claimed in claims 23 and 50, whose rejections have been withdrawn. In claims 23 and 50, the claim clearly states that the second port is coupled to a second functional unit which has commands for performing an arithmetic operation on the row address register and for loading the entire row pointed to by the row address register into a selected set of registers.

57. Regarding applicant's argument that the prior art of record hasn't taught a row deactivate command, the examiner strongly disagrees for the reasons set forth in the rejections above. That is, Farmwald has taught an explicit refresh command. And, Garcia states that "when a memory operation is requested of a DRAM during refresh, the DRAM cannot respond." Hence, it is the

examiner's interpretation that when a refresh is performed, the DRAM is deactivated in the sense that it cannot respond and entertain any additional load/stores. While applicant's invented deactivate command may not perform a refresh, the refresh command reads on the claims because it does deactivate a row during the refresh phase.

58. Regarding applicant's argument that the references fail to show active/inactive register files and commands for toggling between inactive/active states, the examiner strongly disagrees for the reasons set forth above. That is, in Parady, each thread has its own register file. Hence when a first thread is executing, a first register file is active. The remaining threads and register files are inactive. As described throughout Parady, a cache-missing load command is a command which results in a thread switch. Upon a thread switch, a next thread and its register file become active while the previously active thread and register file become inactive. Therefore, a cache-missing load command reads on the claims command for toggling between register file states.

Conclusion

59. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period

will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DAVID J. HUISMAN whose telephone number is (571)272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/David J. Huisman/
Primary Examiner, Art Unit 2183
January 15, 2009

/Kevin L Ellis/
Acting SPE of Art Unit 2187